Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра дослідження операцій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
за спеціальністю 113 «Прикладна математика»
на тему:

Відновлення чіткого зображення по серії розмитих зображень

студента 4 курсу
Кузька Володимира Олександровича

Науковий керівник:
Старший дослідник, кандидат фізико-математичних наук
Рабанович В.І.

Робота заслухана на засіданні кафедри дослідження операцій та рекомендована до захисту в ЕК, протокол № 9 від 23 травня 2023 р.

Завідувач кафедри ДО Іксанов О. М.

Київ-2023

Вступ

Протягом останніх десятиліть, наш світ цифровізується семимильними кроками, зокрема, як і цифровізовується формат людської взаємодії. Зображення, нарівні з іншими видами передачі інформації, як засіб комунікації, стає все більш популярним навколо земної кулі. Вони відіграють досить важливу роль у нашому житті. Фотографії стали не лише засобом зберігання спогадів, проте й доволі потужним інструментом для вираження ідей та комунікації з людьми. Зокрема, фото- та відеодокументування відіграють такі ролі:

- Збереження цінних спогадів.
- Передавання та документування емоцій.
- Вираження мистецтва.
- Аутентифікація та верифікація(наприклад, мобільний застосунок "Дія"активно будується навколо фотопередачі даних).
- Передача публічної інформації(новини) чи конфіденційної інформації.

Окрім цього, наведемо перелік сфер, вклад фотографії в які є критичним:

- Журналістика та засоби масової інформації.
- Медицина.
- Наука та дослідження.
- Криміналістика.
- Маркетинг і реклама.
- Туризм.

Із запровадженням карантину у зв'язку з пандемією CoVid-19, роль фотографії та, зокрема, відеодокументації суттєво змінилася. Варто зазначити, що з розвитком технологій, фотографія почала відігравати ключову річ у військовій сфері, а саме:

- Спостереження з БПЛА та дронів.
- Аналіз підготовчих та тренувальних завдань.
- Розвідка.
- Апостеріорний аналіз бойових завдань.

Левова частка зображень, які фотографуються щосекунди не є надто інформативними: багато з них мають засвічені області або ж навпаки, мають темні області; також фото є розмитими. У цьому дослідженні буде розглядатися проблема нечітких послідовних зображень та буде запропонований алгоритм з відновлення чіткого кадру із серії розмитих фотографій.

Проблема розмитих фото, а саме серії нечітких зображень, найбільш притаманна БПЛА та дронам - починаючи від поривів вітру, перепадів атмосферного тиску, змінної швидкості руху літального апарату, хмарністю, що заважає сфокусуватися на потрібній ділянці.

Нечіткі зображення несуть за собою наступні фактори:

- Втрата виразності та зменшення кількості деталей.
- Погіршення когнітивного сприйняття інформації, переданої зображенням.
- Вплив на емоційну сприйнятливість подальших кадрів.

Серія нечітких зображень, на перший погляд, є великою проблемою. Проте, серія кадрів таки несе в собі певну інформацію. Серед корисних умов для розв'язання даної проблеми варто виділити наступні:

- Сусідні кадри в серії зображень були отримані з мінімальною часовою затримкою.
- Зображення передають одну і ту ж саму картину за виключенням деякої кількості пікселів.

Враховуючи вище зазначені фактори, є підстави припускати, що існує закономірність між сусідніми кадрами, що дозволить дістати потрібну інформацію у вигляді чіткого, сфокусованого зображення.

Розділ 1. Проблема та потенційні застосування. Постановка задачі. Мотивація підходу. Готові програмні рішення.

1.1 Проблема та потенційні застосування

На сьогодні існує багато сфер застосування техніки покадрової зйомки. Зокрема, серія послідовних фотографій може бути отримана шляхом розбиття відеоряду на кадри.

Примітка. Відеорядом вважатиметься така послідовна серія зображень, яка за секунду передає не менше, аніж 24 кадри.

Сфери застосування починаються від відеозйомки з авто- та мототранспорту, закінчуючи аерозйомкою, скажімо з дронів та інших літальних апаратів. Особливість таких фотозйомок полягає в тому, що між сусідніми кадрами майже завжди будуть наявні мікроколивання та інші прояви осциляції, що, у свою чергу, можна інтерпретувати як зсув попереднього зображення на певну кількість пікселів, його поворот на невелику градусну міру та масштабування(розтягування-стиснення зображення відносно попереднього кадру).

Зазвичай, такі коливання не минають безслідно - часто дещо зміщені кадри мають недостатню різкість, і тому візуальний аналіз такого зображення не може принести нової, а деколи взагалі ніякої, інформації.

Очікується, що алгоритм з покращення різкості зображень можна застососувати принаймні у чотирьох випадках:

1. Опрацювання серії зображень апостеріорі. Наприклад, алгоритм обробляє уже відзняту серію зображень, отриману з відеоряду, щоб отримати якісне зображення баобабу, якого туристи спостерігали на фоні африканської савани.

2. Опрацювання серії зображень "на льоту". Наприклад, шукач скарбів використовує літальний дрон для зйомки ландшафту місцевості. На жаль, на відповідній території може переважати вітряна погода, тому через постійні коливання літального апарату та помірну відстань від нього до шукача, людина отримує декілька нечітких кадрів в секунду. В режимі онлайн, алгоритм попарно порівнює сусідні кадри, отримані з дрону, та видає оператору якіснішу версію зображення місцевості.

3. Будь-який випадок із двох вище наведених разом із візуальним аналізом - яке із оброблених зображень краще відображає ділянку фото, яка цікавить оператора.

4. Довільна комбінація до вище запропонованих випадків.

1.2 Постановка задачі

Нехай маємо серію послідовних зображень, що дещо відрізняються один від одного - зсув до 20 пікселів по горизонталі та вертикалі окремо, поворот до 3х градусів, що відбувається відносно центру зображення, різниця розтягнення чи стиснення за відповідним напрямком не перевищує 5 відсотків. Очікується, що іноді зображення матимуть невеликі зміни у фокусувані(масштабуванні) і малий зсув у серії послідовних зображень.

Задача: створити алгоритм що із даної серії зображень видасть якісну картинку, що матиме менші проблеми з фокусом та різкістю або не матиме їх зовсім. Алгоритм повинен бути простим для інтерпретації та таким, що працює не повільніше ніж, наприклад, $1/24$ секунди, щоб заміна неякісного зображення на опрацьоване алгоритмом, яку робить фільтр, відбувалася непомітно для спостерігача.

Очікувані режими роботи алгоритму:

1. Попарне порівняння сусідніх(послідовних) зображень та заміна одного з них якісним зображенням або ж повернення нового якісного зображення як результат роботи алгоритму;

2. Порівняння усієї серії зображень та повернення якісного зображення для наступного аналізу даних з можливістю відтворення дрібних деталей.

1.3 Мотивація підходу

Проживаючи в епоху машинного навчання, можна було б покластися на системи штучного інтелекту, які після фази навчання, могли б так чи інакше впоратися із задачею. Наведемо недоліки такого підходу [4]. Насамперед, для цього потрібно:

- Вимагати наявність бази даних зображень, на яких системи штучного інтелекту могли б вчитися. Частково, проблема обмеженості навчальних даних вирішується шляхом штучного нарощення фотографій: поворот, масштабування, симетрія відносно різних осей та зсув уже наявних зображень.

Проте все ще потрібно мати початкову базу зображень для навчання системи. Окрім цього для тренування результативної моделі машинного навчання потрібно мати різноманітні, представницькі зображення. Зокрема, потрібно вимагати наявність різних типів дефектів на зображені, різних видів розмиття. В іншому випадку, модель навчиться розрізняти та фокусувати лише один або декілька видів дефектів, що при відновленні якості фотографії інколи може призвести до його погіршення.

- Час для навчання системи та, можливо, час на аналіз результатів, покращення системи та її перенавчання в перспективі. Навчання моделі штучного інтелекту - ресурсоємкий процес. Для її хорошого функціонування потрібно провести першу, початкову, фазу навчання - вона забере найбільше часу. Далі, у майбутньому, слід дотреновувати модель новими фотографіями, що раніше не були в системі. Такі картинки мають містити нові види розмиття та деформацій. Навіть наявність вичерпної бази неякісних зображень не робить дотренування моделі непотрібним та зайвим кроком. З розвитком технологій, що відбувається надто стрімко, можуть змінитися ключові концепти фотографії або ж з'являться деталі, що матимуть важливу роль в утворенні зображень. Дотренування моделі - менш ресурсоємкий, проте також вимогливий процес і важливий процес.

- Утворення артефактів. Оскільки логіка, якою керується штучний інтелект, полягає у числах та векторах - результат зміни зображення залежить виключно від них. У випадку неякісного тренування моделі, вона замість покращення зображень може вирішити додати артефакти у область, що розглядається - поставити чорну цятку, або провести світлу лінію. У порівнянні з оригіналом, така зміна лише погіршить якість фотографії.

- Втрата деталей. Аналогічно з попереднім пунктом, модель здатна погіршити якість зображення або окремих його областей, шляхом розмиття певного переліку деталей, зміною їх кольору до кольору фону, або ж, що також зустрічається у моделях машинного навчання - різний ступінь згладжування зображень, що призведе до втрати деталей в цілому.

- Відсутність розуміння семантики фотографій. Системи штучного інтелекту, що можуть застосовуватися для відновлення якості нечітких зображень, не наділені здатністю когнітивного розуміння контексту. Немає гарантії, що вони розумітимуть семантику зображень, які вони оброблятимуть - вони можуть не розрізняти об'єкти, скажімо людину від автомобіля як представників різних класів на фотографії. Також фон зображення може мати декілька ступенів, проте штучний інтелект може їх узагальнити до одного класу. У загальному, це призведе до неточного та неякісного відновлення зображення без збереження усіх деталей та об'єктів.

- Суб'єктивність оцінки відновлення. Оскільки результат якості фото залежить, зокрема, від тренувальної бази - не можна чітко стверджувати що фокус на одному фото матиме такий же успіх на іншому фото. Поміж тим, не зрозумілі функції критерію чи оптимізації, за якими б визначалася результативність роботи систем штучного інтелекту. Ймовірно, чимало покращених фотографій лежать близько до оптимальної точки, проте не можна стверджувати що наступне покращення дозволить зчитати яку-небудь інформацію із зображення - чи то певні деталі, чи то форма об'єкту.

- Неможливість вивчити системою всі фотографії з унікальними деформаціями. Нехай, обробляємо зображення розміром 10×10, що дає сумарну кількість пікселів - 100. Також візьмемо спрощену ситуацію, коли для кожного пікселя є 100 рівнів інтенсивностей, а не 256, як в стандартній моделі зображень. Провівши нескладні підрахунки, отримаємо загальну кількість унікальних зображень, що рівна 100^{100}. Навіть за наявності 1 000 000 серверів, де кожен оброблятиме по 1 000 000 000 зображень в секунду, знадобиться близько $3 \cdot 10^{177}$ років для їх опрацювання.

Вищезгадані недоліки наштовхують на думку, що програмні рішення, що полягають на ідеї машинного навчання, можуть бути поганими для відновлення якості зображень. Вони можуть не забезпечувати достатньої деталізації, точності та відтворення контексту, що може призвести до втрати якості початкового зображення.

Окрім цього, може ховатися наступна невизначеність: проблема трактування роботи нейронної мережі. За умови ручного створення такої системи штучного навчання, існує проблема тлумачення блоків та шарів - основних компонент нейронних мереж, оскільки дані між ними передаються у вигляді чисел та чисельних векторів, що робить задачу інтерпретації досить тяжкою, а деколи – майже неможливою, у когнітивному сенсі.

Для системи типу AutoML, де розробник не створює нейронну мережу - це робить AutoML, корегуючи певні параметри, проблема з інтерпретацією архітектури системи та її поведінки також залишиться. Проте разом із нею додадуться деякі нові аспекти щодо її інтерпретації:

Системи типу AutoML (Automated Machine Learning) можуть бути складними для інтерпретації самих себе і результатів своєї роботи з кількох причин:

- Ефект чорної скриньки. Системи типу AutoML можуть використовувати складні моделі машинного навчання, такі як нейронні мережі або об'єднання моделей. Такий підхід вимагатиме наявність величезної кількості параметрів і складних внутрішніх структур, що ускладнить їх інтерпретацію. Отримані моделі можуть бути непрозорими, тобто не вдасться просто зрозуміти, як саме вони приймають рішення.

- Недостатнє поле для інтепретації. AutoML зазвичай ставить за мету автоматизувати процес побудови моделей, натомість інтерпретація результатів не є основним пріоритетом. У результаті, існує ймовірність, що отримання інтерпретації внутрішньої логіки і рішень, що прийняла модель, стане важким або деколи непосильним завданням. Відсутність пояснювальності може бути проблемою в областях, де важливо зрозуміти причини, які стоять за прийнятими рішеннями.

- Складність функцій оптимізації. Системи типу AutoML використовують автоматичну оптимізацію алгоритмів та їх моделей для знаходження найбільш оптимального набору параметрів для вирішення задачі. Однак, такий процес може бути доволі складним та залежатиме від багатьох факторів, наприклад, вибраний алгоритм оптимізації, обмеження на ресурси, метрика для вимірювання результативності системи. Описана складність може робити тлумачення моделі важким завданням.

- Ефект перенесення. AutoML може використовувати попередньо створені моделі та алгоритми, що були навчені на інших наборах даних для вирішення інших завдань. У таких випадках інтерпретація може значно ускладнитися, оскільки результати можуть відображати залежності та шаблони, що були виявлені на інших наборах навчальних даних, які були призначені для вирішення інших завдань. Така умова може свідчити, що перенесені моделі не завжди зможуть адекватно адаптуватися до нових завдань.

Насамкінець, не існує гарантії щодо швидкодії навченої системи штучного інтелекту під час покращення фокусу зображення - процес може тривати десятки секунд, особливо якщо це фотознімки однорідної поверхні, наприклад лісу чи океану з кабіни літака. Умова швидкого виконання алгоритму є критичною для його застосування "на льоту". Щоправда існує можливість оцінки її швидкодії на основі емпіричних знань: після застосування такого алгоритму на великій вибірці зображень. Це мотивує на створення нового алгоритму, що буде легким для інтерпретації та працюватиме достатньо швидко, щоб застосовувати його "на льоту".

1.4 Готові програмні рішення

Для покращення якості розмитих зображень в програмних рішеннях застосовуються різні ідеї. Ось декілька з них:

- Відновлення за допомогою деконволюції. Даний підхід полягає в теорії оберненої фільтрації, яка дозволяє відновлювати деталі та зменшувати ефект розмиття, якщо з початку відомий вигляд функції розмиття. Наприклад, це можуть бути фільтри розмивання, що дозволять зрозуміти шаблони розмиття, а потім використати його у процесі відновлення зображення.

- Використання теорії ймовірностей та методів математичної статистики. Існує низка готових програмних рішень, які використовують розподіли теорії ймовірностей та статистичні методи для відновлення зображень. Вони можуть аналізувати статистику фотографій, використовувати припущення про розподіл пікселів на них та використовувати цю інформацію для відновлення деталей та поліпшення якості загалом.

- Використання машинного навчання. З приходом ери машинного навчання, відбулося утворення цілого класу програмних рішень, що ґрунтуються на даному концепті. Зокрема, глибоке навчання є популярним напрямком у відновленні розмитих зображень - застосовуються витончені техніки обробки, перетворення та систематизації зображень.

- Пошук ключових точок чи областей, та подальша обробка зображень з їх використанням. Зокрема, знаходження ключових точок або областей дозволяє знаходити головний шаблон розмиття на фото, що, в подальшому, робить покращення якості розмитих зображень більш досяжним завданням.

На сьогодні, найбільш популярними та ефективними програмними рішеннями для відновлення якості розмитих зображень є наступні:

- Adobe Photoshop. Це один із найбільш популярних графічних редакторів на сьогодні, який має вбудовані інструменти для поліпшення якості фотографій, зокрема, функцію відновлення розмитих зображень. Він пропонує фільтри та інструменти для усунення розмивання і покращення деталей.

Наприклад, програмне забезпечення використовує різні області для порівняння ступеню розмиття[2]. Після, цього застосовується вбудована функція 'Усунення тремтіння' , 'Деталізація' або 'Зменшення артефактів'.

- SmartDeblur. Ця програма використовує поєднання деконволюції та водночас використовує статичні методи. Зокрема, аналізується зображення на шаблон зміщення[3], робиться відповідне припущення про розподіл пікселів на зображенні. Після цього, відбувається деконволюція шляхом оберненого перетворення Фур'є. Зокрема, використовуються такі регуляризуючі інструменти: фільтр Вінера, регуляризація Тихонова.

- Focus Magic: Ця програма пропонує користувачам покращення якості фотографій, що піддалися розмиттю; дозволяє відновити їх деталі, їх чіткість за допомогою статистичних методів. На відміну від попереднього, розробник не розголошує на якому принципі будується програма. Натомість, вони пропонують 5 видів покращення якості зображення

  - Auto-focus.
  - Focus.
  - Fix motion blur.
  - Increase resolution x2.
  - Increase resolution x4.

- DxO PhotoLab: PhotoLab є інтегрованим редактором для обробки фотографій. Він має інструменти для усунення розмивання, покращення деталей та відновлення якості зображень, покращення різкості.

- Skylum Luminar. Це фоторедактор, що пропонує набір інструментів для ретуші та поліпшення фотографій. Зокрема, розробники пропонують функцію з відновлення розмитих зображень та підвищення їх чіткості. Програмне забезпечення використовує технології штучного інтелекту для покращення деталей та відновлення якості.

- DeblurMyImage: Ця програма пропонує інструменти для відновлення розмитих фотографій, використовуючи статистичні методи деконволюції та інші алгоритми для поліпшення якості зображень.

- Topaz Sharpen AI: Це програмне рішення, яке використовує штучний інтелект для відновлення розмитих зображень та покращення їх чіткості та деталізації. Воно використовує алгоритми машинного навчання, щоб автоматично виявляти та відновлювати розмиття на зображеннях. Доступні три режими роботи програми:

  - Тремтіння камери та фотографії в русі.
  - Фокусування.
  - Згладжування.

Варто зауважити, що вище перелічене програмне забезпечення вирішує лише одну із підзадач поставленої проблеми. Вони фокусують лише задане розмите зображення, проте існує ймовірність, що сусідні кадри міститимуть нову, унікальну інформацію. Також, через певні види деформацій, конкретне зображення може мати неправильні пропорції головних об'єктів.

Розділ 2. Наївний підхід

2.1 Ідея

Було вирішено провести перевірку припущення, яке полягала в тому, що мікроколивання глобально не впливатимуть на зображення, можливо, постраждає інформація на краях. Отже, фотографію із даної серії можна ототожнити до вибраного кандидатита, з яким далі проводитиметься обробка з надання зображенню різкості.

Було вирішено одразу перевірити гіпотезу наступним чином. Довільне чітке зображення зміщується на a пікселів праворуч по горизонталі та b пікселів вниз по вертикалі, $a, b \in \{0, 1, 2, 3\}$. Після цього обраховується модуль різниці зображень, щоб перевірити гіпотезу.

2.2 Результати та аналіз

Результати експериментів показали, що припущення хибне: так званий 'білий шум' був присутній на всьому зображенні, лише контури лампи розжарювання, дерев'яної полички та тонкого кабелю свідчили про обриси попереднього фото(див. зображення 2.2).

Було вирішено подивитися на відхилення чисельно, а саме: побудувати гістограму метрики відхилення за стовпчиками зображення. Метрика рахується наступним чином:

1. Для кожного стовпця рахується сума за всіма пікселями та усіма каналами(RGB - Red, Green, Blue). Максимальне значення відхилення для кожного стовпця дорівнює $\delta = N * 255 * 3$, де N - висота зображення.

2. На початку та кінці додаються по 2 стовпчики із нульовим відхиленням. Така дія дозволить коректно проводити операцію із вікном moving average.

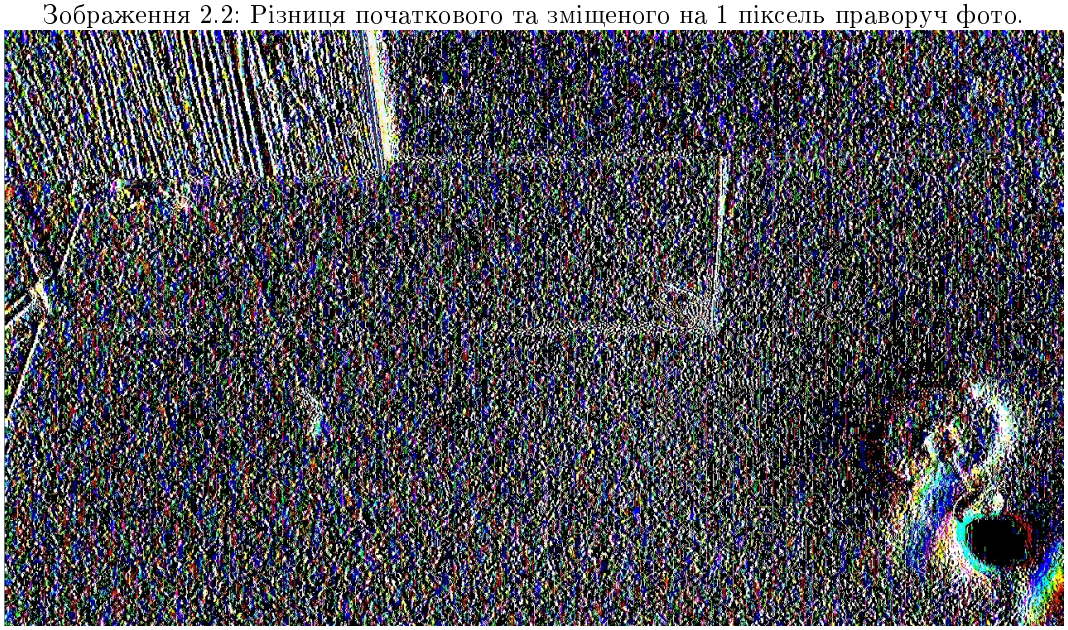
3. Далі, відхилення кожного стовпчика ділиться на суму відхилень вікна moving average із шириною 5. При цьому, ділення відбувається для центрального елемента вікна.

   Усі нормування відбуваються незалежно, тобто нормоване значення n стовпчика за 4-ма сусідніми не вплине на нормування $n + 1$ стовпчика.

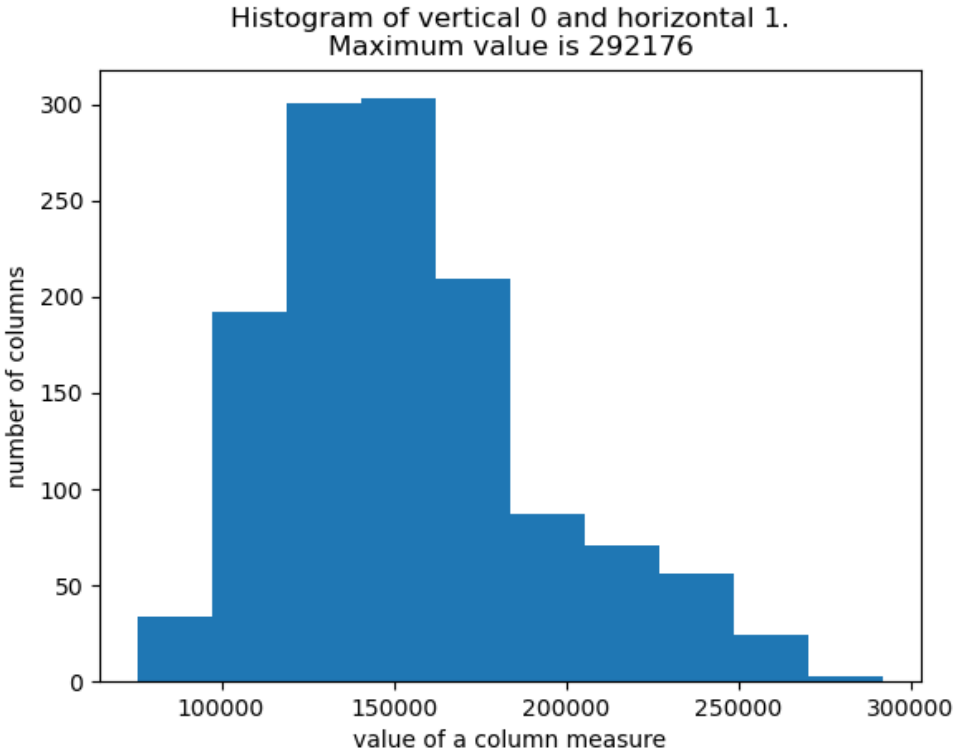
Зображення 2.1: Початкове фото.

Проілюстроване зображення має розмірність 1280×720, що дає кількість стовпчиків - 720.

Зображення 2.2: Різниця початкового та зміщеного на 1 піксель праворуч фото.



Зображення 2.3: Гістограма різниці початкового та зміщеного фото.



Розділ 3. Алгоритм

3.1 Короткий опис алгоритму.

Нехай маємо серію розмитих зображень однакових розмірів, що відрізняються між собою мікроколиваннями у вигляді повороту зображення, його зміщення по вертикалі та(або) горизонталі, та масштабування. Робота алгоритму відбуватиметься, з використанням двох зображень, попарно порівнюючи сусідні зображення на кожній ітерації.

Більшу частину алгоритму складатимуть кроки, що максимально усуватимуть коливання між сусідніми зображеннями, зокрема, створимо оператор, що перетворюватиме перше зображення з пари в друге.

Розглянемо роботу алгоритму.

1. Кожне зображення розбиваємо на 4 рівних частинки.
2. У кожній із четвертинок визначаємо особливі точки - вони слугуватимуть індикатором того, яким чином відбувається перетворення одного зображення у інше. Отримали по 4 локальних особливих точки на кожному зображенні.
3. У кожному із зображень рахуємо глобальну особливу точку як середнє відповідних локальних особливих точок.
4. Будуємо локальні та глобальні вектори, де початком векторів слугуватимуть відповідні точки із першого зображення, а кінцем - точки із другого зображення. Глобальний вектор називатимемо вектором зсуву $\overrightarrow{S}$.
5. Обираємо два вектори чотирьох локальних векторів - $\overrightarrow{\tilde{A}A_1}$ та $\overrightarrow{\tilde{B}B_1}$.
6. Зміщуємо точки $\tilde{A}$ та $\tilde{B}$ на вектор $\overrightarrow{S}$ і отримуємо точки A та B.
7. Створюємо вектори $\overrightarrow{OA}$, $\overrightarrow{OA_1}$, $\overrightarrow{OB}$ та $\overrightarrow{OB_1}$, що будуть використовуватися для побудови оператора трансформації. Визначаємо $\overrightarrow{OA}$ та $\overrightarrow{OB}$ як базисні вектори простору трансформацій. Тут A та B - особливі точки із першого зображення, зміщені на вектор $\overrightarrow{S}$, A_1 та B_1 - особливі точки із другого зображення; точка O - центр зображення.
8. Конструюємо оператор трансформації L, як матрицю координат векторів $\overrightarrow{OA_1}$ та $\overrightarrow{OB_1}$ у базисному просторі.
9. Визначимо новий базис, щоб він співпадав із початковим базисом, зміщеним його початком до центру зображення - одиничні координати мають довжину один піксель та лежать горизонтально праворуч та вертикально донизу.
10. Створюємо матриці переходу із нового до старого базису, та робимо перехід матриці L до нового базису.
11. Шукаємо обернену матрицю афінного перетворення зображення та виконуємо його для другого зображення - отримуємо перше зображення.
12. Далі, для обох зображень визначаємо найбільші об'єкти та порівнюємо їх площу. Додаємо ці записи в загальну вибірку усіх опрацьованих зображень. Порівнюємо площу особливих об'єктів на усіх кадрах, оброблених алгоритмом, і обираємо кандидата на фокусування, де площа є медіаною даної вибірки.
13. Фокусуємо обране розмите зображення за допомогою оберненого перетворення Фур'є.

3.2 Особливі точки. Центр мас зображення.

Під час повороту, зсуву, чи розтягнень зображень бажано знайти таку точку, яку буде легко ідентифікувати після згаданих перетворень. Серед варіантів вибору такої особливої точки, було вирішено зупинитися на центрі мас зображення, ідею якого запозичено з механіки.

Обрахування x-координати такої точки відбувається за наступним алгоритмом.

Нехай маємо матрицю інтенсивностей пікселів RGB зображення B розміром $w \times h \times 3$, де w та h – ширина та висота зображення відповідно. Обираємо один із кольорових каналів, та позначимо одноканальну матрицю пікселів як b. Визначимо вектор-рядок $\tilde{x}$ та $\tilde{y}$:

$$\tilde{x} = \begin{pmatrix} 1 & 2 & .. & w \end{pmatrix} \tag{3.1}$$

$$\tilde{y} = \begin{pmatrix} 1 \\ 2 \\ .. \\ h \end{pmatrix} \tag{3.2}$$

X-координата обчислюватиметься за наступною формулою:

$$x = \frac{\sum_{m=1}^{h} \sum_{k=1}^{w} \tilde{x}_k \cdot b_{m,k}}{\sum_{m=1}^{h} \sum_{k=1}^{w} b_{m,k}} \tag{3.3}$$

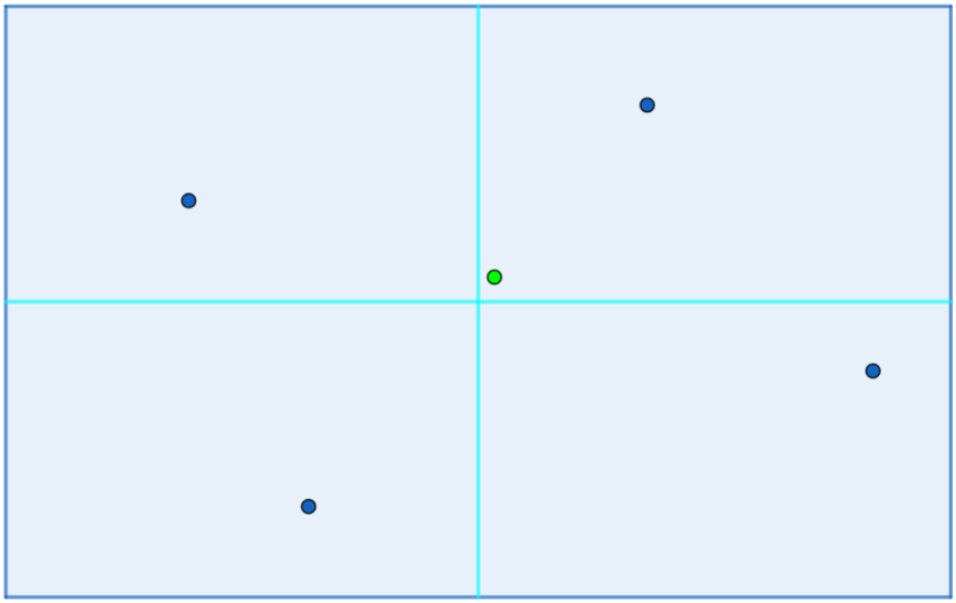
Y-координата вираховується аналогічно:

$$y = \frac{\sum_{m=1}^{w} \sum_{k=1}^{h} \tilde{y}_k \cdot b_{k,m}}{\sum_{m=1}^{h} \sum_{k=1}^{w} b_{m,k}} \tag{3.4}$$

Після цього відповідні координати заокруглюємо до найближчого значення.

Далі, ділимо зображення на 4 рівних частини, і в кожному зображенні вираховуємо центр мас маленьких зображень. Також, вираховуємо центр мас всього зображення як середнє центрів мас маленьких зображень.

Зображення 3.1: Знаходження п'яти особливих точок.



3.3 Вектори трансформації

Вектори трансформацій утворюються наступним чином.

1. У першому зображенні вираховуємо 5 особливих точок: 4 - центри мас маленьних зображень та їх середнє.
2. Таку ж операцію виконуємо для другого зображення.
3. Поєднуємо відповідні центри маси двох зображень у вектори: початком всіх векторів слугуватимуть центри мас першого зображення, кінцем - центри мас другого зображення.

Отримали 4 вектори трансформацій для маленьких зображень - далі називатимемо їх локальними векторами, та вектор зображення загалом, вектор зсуву - глобальний.

Зображення 3.2: Суміщення відповідних особливих точок.

3.4 Лінійність оператора.

Вважаємо, що допустимі операції, що спричиняють різні кадри у серії зображень - це малий зсув, поворот та масштабування(розтягування-стиснення) - є приблизно лінійними операціями, а отже оператор трансформації зображень також є лінійним афінним перетворенням.

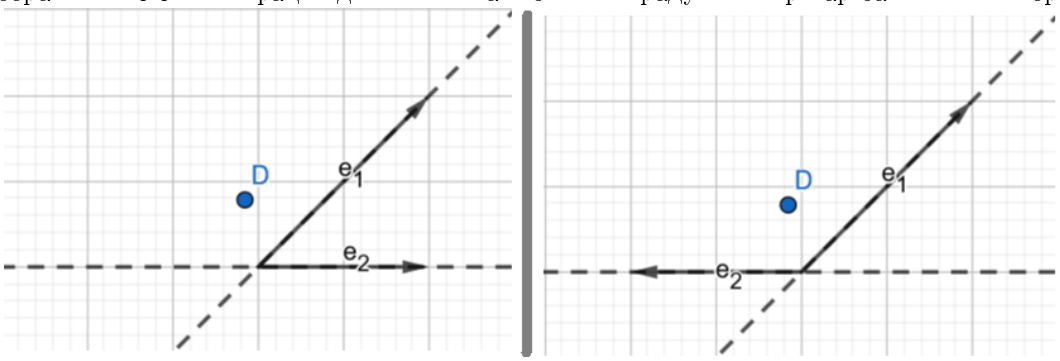
3.5 Базисні вектори. Інформативність пари базисних векторів

Оператор перетворення від першого зображення до другого буде реалізовуватися на парі локальних векторів - базисних векторів. Серед усіх можливих $\frac{4!}{(4-2)!} = 12$ комбінацій, потрібно обрати таку пару, елементи якої найменш залежать одне від одного. Зокрема, пара базисних векторів, які утворюють кут 3° внесе більше спотворення у процес відновлення зображення, аніж пара базисних векторів, які утворюють кут 45°.

Оскільки найкращий випадок - пара векторів, що утворюють 90°, будемо шукати таку комбінацію локальних векторів, різниця 90° та їх градусної міри найменша.

Дві пари векторів, що мають однакову різницю між їх градусними мірами та 90°, але лежать по різні боки від 90°, будемо вважати еквівалентими базисними парами векторів у сенсі спотворення алгоритму. Це буде виконуватися, оскільки одна із чвертей, у якій лежить конкретна точка в базисі першої пари, відповідає іншій чверті в базисі другої пари векторів. **Примітка.** У даному прикладі, чвертю вважатимемо одну із чотирьох підплощин, на які розбивають площину пара базисних векторів.

Зображення 3.3: Ілюстрація ідентичних за схожістю градусних мір пар базисних векторів.



3.6 Створення оператору перетворення

Оскільки оператор L переводить перше зображення в друге, це означає, що $L(\overrightarrow{OA}) = \overrightarrow{OA_1}$ і $L(\overrightarrow{OB}) = \overrightarrow{OB_1}$.

Перейдемо до одиничних векторів по осях OA та OB. Позначимо $\overrightarrow{e_1} = \frac{\overrightarrow{OA}}{|\overrightarrow{OA}|}$, $\overrightarrow{e_2} = \frac{\overrightarrow{OB}}{|\overrightarrow{OB}|}$. Визначимо дану пару векторів $(\overrightarrow{e_1}, \overrightarrow{e_2})$ як базис.

З лінійності оператора L випливає, що $L(\overrightarrow{e_1}) = \frac{\overrightarrow{OA_1}}{|\overrightarrow{OA}|}$ і $L(\overrightarrow{e_2}) = \frac{\overrightarrow{OB_1}}{|\overrightarrow{OB}|}$.

Зауважимо, що саме результат перетворення одиничних базисних векторів оператором L покаже його явний вигляд, а саме:

$$L(\overrightarrow{e_1}) = \frac{\overrightarrow{OA_1}}{|\overrightarrow{OA}|} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, L(\overrightarrow{e_2}) = \frac{\overrightarrow{OB_1}}{|\overrightarrow{OB}|} = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \tag{3.5}$$

Оскільки базисні вектори будуть не обов'язково перпендикулярні, то координати $(x_1, y_1), (x_2, y_2)$ будемо шукати шляхом паралельного перенесення осей OA та OB на кінці векторів $\frac{\overrightarrow{OA_1}}{|\overrightarrow{OA}|}$ і $\frac{\overrightarrow{OB_1}}{|\overrightarrow{OB}|}$ почергово. Далі знаходимо потрібні координати як перетини паралельно перенесених OA та OB з початковими, базисними осями.

Отримали вигляд оператора перетворення L в базисі $\langle \overrightarrow{e_1}, \overrightarrow{e_2} \rangle$.

$$L_{\langle \overrightarrow{e_1}, \overrightarrow{e_2} \rangle} = \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \tag{3.6}$$

3.7 Зміна базису

Щоб мати змогу зручніше оперувати матрицею перетворення L, потрібно перевести його у базис, що буде відповідати початковій парі векторів, лише зміщеній на центр зображення.

Для цього скористаємося формулою переходу матриці до нового базису:

$$L_{\langle \overrightarrow{E_1}, \overrightarrow{E_2} \rangle} = T L_{\langle \overrightarrow{e_1}, \overrightarrow{e_2} \rangle} T^{-1}, \tag{3.7}$$

- $\langle \overrightarrow{E_1}, \overrightarrow{E_2} \rangle$ - базисні вектори початкової системи координат. Їх довжина рівна одному пікселю. Перший вектор $\overrightarrow{E_1}$ напрямлений горизонально праворуч, другий, $\overrightarrow{E_2}$, перпендикулярно донизу.
- T - матриця переходу з базису $\langle \overrightarrow{E_1}, \overrightarrow{E_2} \rangle$ в базис $\langle \overrightarrow{e_1}, \overrightarrow{e_2} \rangle$.

Зокрема, матриця T - це матриця, стовпці якої - координати розкладу векторів $\langle \overrightarrow{e_1}, \overrightarrow{e_2} \rangle$ у базисі $\langle \overrightarrow{E_1}, \overrightarrow{E_2} \rangle$:

$$\overrightarrow{e_1} = \alpha_{11}\overrightarrow{E_1} + \alpha_{12}\overrightarrow{E_2} \tag{3.8}$$

$$\overrightarrow{e_2} = \alpha_{21}\overrightarrow{E_1} + \alpha_{22}\overrightarrow{E_2} \tag{3.9}$$

Отже, оператор переходу до нового базису T матиме наступний вигляд.

$$T = \begin{pmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{pmatrix} \tag{3.10}$$

3.8 Обернене афінне перетворення. Зміщення зображення на глобальний вектор зсуву.

Враховуючи, що перетворення, які описує оператор L, а саме зміщення, поворот та розтягування-стиснення, є афінним перетворенням. Отже, щоб повернути друге зображення до початкового, будемо використовувати обернене афінне перетворення.

Матриця афінного перетворення має вигляд:

$$M = \begin{pmatrix} L_{11} & L_{12} & S_1 \\ L_{21} & L_{22} & S_2 \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.11}$$

- L - обчислений оператор перетворення.
- S - вектор зсуву по осі oX та oY відповідно.

Оскільки побудова оператора явно не враховує глобальний вектор зміщення, ми підставляємо його як вектор S у матрицю афінного перетворення.

Далі, шукаємо M^{-1} та виконуємо обернене перетворення другого зображення.

У випадку, коли оберненої матриці до M не існує, це може свідчити про те, що зображення не відрізняються або ж різниця зображень менша за машинний нуль. В такому випадку, обернене перетворення не відбувається.

3.9 Різниця

Взяття модулю різниці є допоміжним кроком, і не впливає на подальшу роботу алгоритму, а є лише інформативним.

3.10 Обрання кандидата на фокусування із серії попарних порівнянь.

Варто зауважити, що цей крок є дуже важливим, оскільки алгоритм апріорі не може знати яке із зображень передає правильні пропорції об'єктів(стиснені-розширені), а зважаючи на постановку задачі, такі послідовні зміни можуть призвести до суттєвої різниці. У алгоритм закладено припущення, що різниця між стисненням-розширенням між сусідніми кадрами не може перевищувати 5%. Отже, навіть на вибірці у 100 зображень, де кожний наступний кадр має розтягнення 1.02 у відношенні до попереднього кадру, призведе до різниці розмірів об'єктів у $1.01^{100} \approx 7.2$ разів. Це можна уявити, наприклад, як серію зображень, де розміри машини на першому та останньому кадрі різняться у 7.2 разів. Отже, вибір зображення із правильними пропорціями об'єктів на них є важливим крокок алгоритму.

Ідея обрання зображення із усередненою площею найбільших об'єктів ґрунтується на законі великих чисел - а саме величезна кількість кадрів, які оброблятиме алгоритм, створить нормальний їх розподіл(у розумінні деформації об'єктів) на великій вибірці та забезпечить в середньому(або майже завжди) хорошу роботу алгоритму.

На сьогодні, професійні камери навіть на 10-секундному часовому відрізку можуть забезпечити алгоритм близько 10 000 кадрів. Тому, сучасні технічні можливості фото- та відеоапаратури вирішують проблему з наявністю великої вибірки та утворення ними нормального розподілу.

Для визначення найбільших об'єктів на зображенні, зокрема, їх контурів, будемо користуватися алгоритмом [1], що ґрунтується на топологічному аналізі бінарного(чорно-білого) зображення. В його основі лежить деревовидна структура, яка аналізує об'єкти зображення та визначає їх контури.

Я не буду безпосередньо реалізовувати даний алгоритм, оскільки він входить в бібліотеку з обробки та роботою із зображеннями OpenCV.

3.11 Фокус

Розглянемо за яким принципом отримується розмите зображення із чіткого.

$$I * K + N = B, \tag{3.12}$$

- I - Початкове сфокусоване зображення.
- K - Ядро розмиття(для прикладу розглядатимемо гаусівське).
- $*$ - Операція згортки.
- N - Шум, що проявляється у випадкових високочастотних коливаннях.
- B - Отримане розмите зображення.

Посилаючись на нормальний розподіл деформації об'єктів на великій вибірці кадрів, отримаємо, що з характеру деформації фотографій випливатиме характер їх розмиття. Вважатимемо, що на великій вибірці розмиття фото можна описати за допомогою Гаусівського розмиття. З цих міркувань, використовуватимемо відповідне ядро для фокусування розмитих зображень.

Зокрема, для спрощеного вигляду попереднього рівняння без врахування шуму, перетворення Фур'є обертає згортку на операцію їх множення:

$$F(I) \cdot F(K) = F(B), \tag{3.13}$$

- F - перетворення Фур'є.

Щоб врахувати шум, скористаємося, зокрема, фільтром Вінера [7].

$$F(I) \cdot F(K) \cdot \left(1 + \frac{SNR}{\mid F(K) \mid^2}\right) = F(B), \tag{3.14}$$

- SNR - співвідношення сигнал\шум зображення.

Для фокусування зображення скористаємося оберненим перетворенням Фур'є.

$$I = F^{-1}(F(I)) = F^{-1}\left(\frac{F(B)}{F(K)} \cdot \left[\frac{1}{1 + \frac{SNR}{|F(K)|^2}}\right]\right) \tag{3.15}$$

- F^{-1} - обернене перетворення Фур'є.

Оскільки трьохканальні зображення (RGB) напряму не можна обробити перетворенням Фур'є, тому будемо окремо працювати з кожним каналом та потім суміщати результат з кожного каналу у кольорове зображення.

Розділ 4. Результати

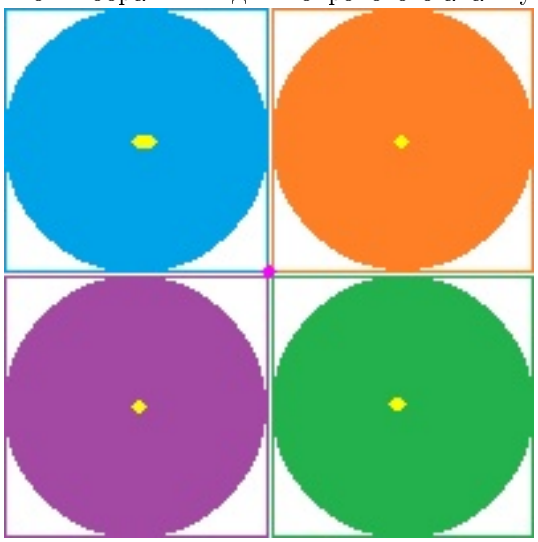
4.1 Перші результати. Невідповідність прямого та оберненого перетворення на практиці.

Перші тести проводилися для перевірки роботи оператора перетворення, а саме чи переходить друге зображення в перше після його оберненого афінного перетворення. На ввід подавалися пари чітких зображень: початкове та зміщене на декілька пікселів вниз і праворуч і повернуте до 2 градусів оригінальне зображення.

З роботи алгоритму було помічено, що під час дії оберненого оператора на друге зображення із пари кадрів, воно стає більш схожим до початкового, але не робить це остаточно. Зокрема, поворот по градусах не відбувається в початкове положення.

Було вирішено провести покроковий аналіз алгоритму, перевіривши його роботу на зображеннями із декількома деталями - 4 кола у кожній із четвертей. Під час перевірки було виявлено певну закономірність і у зв'язку з цим було запропоновано наступну гіпотезу.

Зображення 4.1: Тестове зображення для покрокового аналізу роботи алгоритму.



Зображені крапочки - це 4 локальних вектори, які характеризують поворот та зсув. Середнє цих точок, що зображена фіолетовим кольорим, лежить по центру зображення на перетині чотирьох ліній.

4.2 Гіпотеза №1. Поведінка на різних розмірностях зображень.

У серці комп'ютерної обробки зображень лежить правило: координати країв пікселей - це завжди цілі числа. Проте, аналізуючи роботу оператора та інших застосувань афінного перетворення фотографій, можна прослідкувати, що часто такі трансформації переводять пікселі не в квадрати, а у паралелепіпеди, чи прямокутники, не зберігаючи рівність їх сторін. У результаті, краї таких чотирикутників вже не мають цілих координат, що суперечить закону про їх цілочисельність. У такому випадку відбувається заокруглення координат чотирикутників у найближчий бік, часто ще більше спотворюючи початкове зображення.

Вплив заокруглення найбільш помітний на зображеннях, де відношення площі округлення до площі всього зображення найбільша, тобто, на фотографіях із малою роздільною здатністю. Це має місце, оскільки площа заокруглення не перевищує одного пікселя, а площа всього зображення залежить від кількості усіх пікселей. Отже, найбільш вразливими до операції заокруглення координат пікселів під час їх перетворень є зображення із малим розширенням. Зважаючи на цей факт, було висловлено припущення, що зображення із високою роздільною здатністю менше піддаватимуться впливу заокруглення, а отже, результат роботи оберненого афінного перетворення буде кращим.

Для перевірки гіпотези було взято 3 зображення:

- $200 \times 200 = 40\ 000$ пікселей (4 кола).
- $1920 \times 1080 = 2\ 073\ 600$ пікселей .
- $3840 \times 5760 = 22\ 118\ 400$ пікселей.

Ці зображення були повернуті на однакову градусну міру і зсунуті на однакову кількість пікселів.

Результати оберненого перетворення свідчили протилежне: припущення виявилося хибним - на менших зображеннях алгоритм працював майже ідеально - поворот відбувався в початок, залишалося відкорегувати лише зміщення зображення; на зображеннях середнього розміру - алгоритм повертав фото, проте не остаточно; на великих зображеннях - алгоритм майже не працював, ба більше, він розвертав його на частку градусу в інший бік, намагаючись розширити фото на чорні області.

4.3 Метод визначення границь об'єкта(різка зміна кольору).

Разом із тим, з'явилося цікаве спостереження, яке найкраще проглядалося на великих зображеннях: різниця неідеального оберненого перетворення залишало досить чіткі контури об'єктів, зокрема автомобіля по центру. Емпірично було визначено, що найбільш чітко визначаються контури тих об'єктів, що лежать ближче до центру. Об'єкти, що лежать далі від центру мали товстіші контури. Це можна легко пояснити тим, що пікселі по центру, повертаючись на певну градусну міру, проходять менший шлях, аніж ті, що знаходяться на краях зображення. Враховуючи наявність похибки під час оберненого перетворення, отримаємо точніший результат при визначенні країв об'єктів, що лежать ближче до центру.

Отже, ми експериментально отримали новий спосіб для визначення країв об'єктів по центру зображень, контури яких відділяють різку зміну кольору.

Зображення 4.2: Різниця застосування оберненого оператора перетворення та початкового зображення.

4.4 Гіпотеза №2. Невідповідність центрів мас.

Наступна гіпотеза, що може пояснити недосконалий результат полягає у тому, як ми визначали особливі точки, обравши їх за центри мас чотирьох рівних частин зображення. Проблема з'являється коли початкове зображення зміщується чи повертається - тоді у в одному із чотирьох віконечок зображення з'являються нові пікселі. На невеликих зображеннях (приблизно до півмільйона пікселів) під час повороту чи зміщення з'являється доволі небагато пікселів, що несуть нову інформацію. На зображеннях із більшим розширенням при такому ж повороті чи зміщенні з'являється більша кількість нової інформації, що зміщує центр мас на четвертинках зображення ще далі від початкового, аніж це відбувається на невеликих зображеннях.

4.5 Швидкодія алгоритму.

Будемо окремо оцінювати швидкодію алгоритму до фокусування, та саму операцію фокусування, оскільки, в основному алгоритм лише збиратиме інформацію про площу найбільших об'єктів, та після того – фокусуватиме декілька обраних кандидатів.

Оскільки ця робота носить дослідницько-експериментальний характер, для зручності було обрано інтерпретовану мову програмування Python, що апріорі повільніше за компільовані мови.

Отже, результати наступні – утворення оператора трансформації, обернене перетворення та визначення площі об'єктів займає в середньому від $\frac{1}{25}$ до $\frac{1}{6}$ секунди, що еквівалентно опрацюванню 6-25 зображень за секунду.

Для швидкодії було обрано швидку версію перетворення Фур'є. Варто зауважити, що для кожного зображення обчислювалося по три швидких перетворень Фур'є та їх обернених - для кожного каналу (RGB) по одному.

Результат таких перетворень показав число від $\frac{1}{2}$ до 1 секунди для різних зображень. Спершу, вирішено оптимізувати такий процес та запустити три функції на паралельне виконання, що б теоретично зменшило час роботи утричі. Проте, як виявилося з реалізації швидкого перетворення Фур'є, у бібліотеці numpy – окремий запуск такого перетворення уже оптимізовується на прикладному рівні, використовуючи можливості комп'ютера, що доступні. Отже, розпаралелення 3-х перетворень Фур'є не покращили швидкодію алгоритму.

Перехід та переклад даного алгоритму на компільованій мові програмування, очікувано має принести покращення швидкодії у 2-10 разів [5],[6].

Також, наявність потужнішого процесору або ж трьох машин(процесорів), дозволила б зменшити час фокусування втричі.

4.6 Порівняння результатів із іншими програмними рішеннями

У даному розділі ми будемо порівнювати лише останній крок алгоритму - фокусування, оскільки згадані у першому розділі програмні рішення не призначені для повного вирішення поставленої задачі. Маємо, що обернене перетворення Фур'є застосовувалося на кожен канал окремо, а потім результати поєднувалися у кольорове(RGB) зображення шляхом накладання одного шару пікселів на інший, результат вийшов неприродний, а, зокрема, затемненний. Було вирішено додатково висвітлити зображення. У додатку можна побачити чорно-біле та кольорове відновлене зображення.

Focus Magic фокусував розмите зображення у двох режимах - "Focus"та "Increase Resolution x4". Результати обох режимів є згладжені зображення. Час їх роботи - 20 секунд та 12 хвилин відповідно. Різниця у часу не відповідає різниці у якості.

Topaz Sharpen AI фокусує зображення в середньому за 15 секунд, можна регулювати бажаний рівень подавлювання шуму та рівень розмиття. Результат кращий, аніж в попереднього ПЗ, зображення можна описати як більш деталізоване та різке. Результат можна спостерігати в додатку.

Smart Deblur працював близько 20 секунд, у режимі аналізу розподілу пікселів. Після цього програма видає модель розмиття(у вигляді його сліду) та відновлене зображення. Результат можна побачити у додатку - він вийшов розмитим по краях зображення, проте дещо різким у центральній його частині.

Отже, запропонований нами алгоритм працює в рази швидше, але і передає дещо менше деталей. У той же час не можна вважати різкість, що пропонують готові програмні рішення ідеальною - вона в певному ступені надмірна, штучно створючи та пересвітлюючи краї незначних деталей.

Зображення 4.3: Відновлене та висвітлене кольорове зображення створеним алгоритмом.

Зображення 4.4: Відновлене зображення за допомогою Focus Magic.

Висновки

Основними результатами роботи є:

- Новий алгоритм, що дозволяє перетворити і зблизити незначно деформоване зображення до початкової фотографії.

- Алгоритм, що із серії близьких зображень дозволяє обрати усереднене більш інформативне зображення та частково сфокусувати його.

- Швидкодійний алгоритм, що дозволяє в деяких випадках визначити контури об'єктів на зображенні, без застосування згорток, фільтрів та систем ідентифікації. Така швидкість полягає, зокрема, у лінійному операторі перетворення та застосуванню афінного перетворення пікселів зображення.

Що рекомендується допрацювати:

- Алгоритм із знаходження особливих точок, що дозволить точніше визначати їх навіть після деформації зображення. Таке покращення зробить результат оберненого перетворення зображення більш успішним.

- Розглянути альтернативні методи фокусування зображень, які були б не повільніші за наведений у роботі.

- Розглянути підхід з фокусування зображень з використанням методів теорії ймовірностей, зокрема визначати розподіл пікселів на зображенні, знаходити шаблон розмиття. Очікується, що дана ідея значно покращить результат фокусування.

- Переписати алгоритм на компільованій мові програмування та оптимізувати його, як з боку розпаралелювання та інших прикладних інструментів, так і з теоретичного боку.

- Допрацювати алгоритм, що висвітлюватиме сфокусоване зображення, оскільки при практичному застосування результуюче зображення темніше оригіналу.

Література

[1] Satoshi S. Topological Structural Analysis of Digitized Binary Images by Border Following [Електронний ресурс] / Suzuki Satoshi // Computer vision, graphics, and image processing. – 1985. – Режим доступу до ресурсу: https://www.nevis.columbia.edu/ vgenty/public/suzuki_et_al.pdf.

[2] Зменшення ефекту розмиття від тремтіння камери [Електронний ресурс] – Режим доступу до ресурсу: https://helpx.adobe.com/ua/photoshop/using/reduce-camera-shake-induced-blurring.html.

[3] Fix Blurry and Defocused photos with SmartDeblur [Електронний ресурс] – Режим доступу до ресурсу: http://smartdeblur.net/.

[4] Aurelien G. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems [Текст] / Geron Aurelien., 2019. – 510 с. – (2). – ISBN 978-1-492-03264-9.

[5] Comparative Analysis Of Compiler Performances And Program Efficiency [Електронний ресурс] // Preprints. – 2019. – Режим доступу до ресурсу: https://www.preprints.org/manuscript/201909.0322/v1

[6] FFT performance using NumPy, PyFFTW, and cuFFT [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.johnaparker.com/blog/fft_2d_performance.

[7] Haykin S. Adaptive Filter Theory [Текст] / Simon Haykin., 2013. – 912 с. – (5). – ISBN 978-0132671453.

[8] Лінійна алгебра та аналітична геометрія: Навч. підручнник [Текст] / Рудавський Ю.К, Костробій П.П., Луник Х. П., Уханська Д.В. - Л. : Бескид Біт, 2002. - 262 с.

[9] Computer Graphics Principles and Practice [Текст] / James. D. Foley, Andries van Dam, Steven K. Feiner та ін. - Addison Wesley, 2013. - 1263 с. - ISBN 978-0-321-39952-6.

Додаток А. Результати роботи алгоритмів.

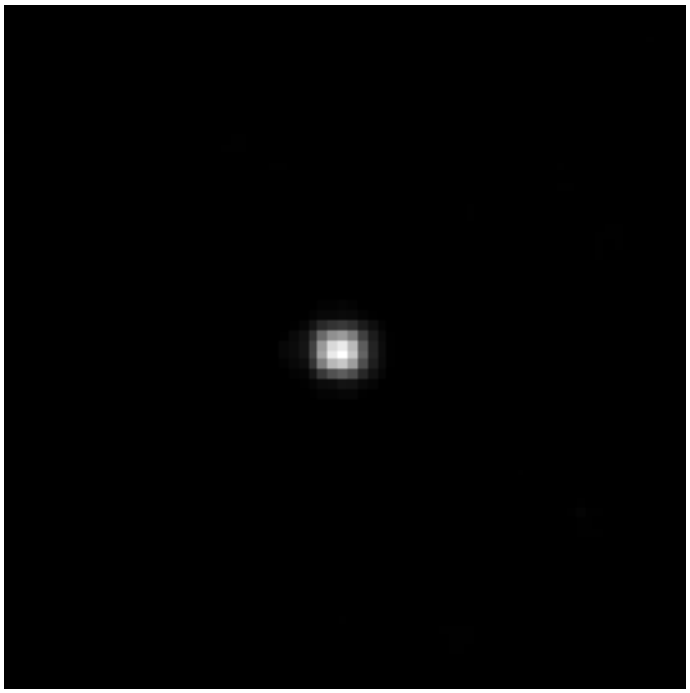
Зображення 4.5: Початкове нечітке зображення.

Зображення 4.6: Відновлене одноканальне(чорно-біле) зображення за допомогою алгоритму.

Зображення 4.7: Відновлене кольорове невисвітлене зображення за допомогою алгоритму.

Зображення 4.8: Відновлене зображення за допомогою Topaz Sharpen AI.

Зображення 4.9: Відновлене зображення за допомогою Smart DeBlur.



Зображення 4.10: Слід розмиття, який визначив Smart DeBlur.

Додаток Б. Код програми.

Повний програми розміщений на хмарному ресурсі GitHub: https://github.com/83demon/dyplom . Наведемо деякі його фрагменти.

Функція, яка обирає пару векторів, що утворює найближчу до 90° градусну міру.

```
def _select_A_and_B(self):
  """Selects two vectors, which will form an angle (BOA) closest to the 90 degrees.
  B = B_tilda + O_O_tilda
  A = A_tilda + O_O_tilda"""
  assert len(self.vectors)>=2
  angles_map = {} #  key is pair of indexes of vectors, value: pair: 1st is how far
                                        the angle from 90 degrees is,2nd is real value
  for i in range(len(self.vectors)):
    for j in range(len(self.vectors)):
      if i!=j and (i,j) not in angles_map.keys() and (j,i) not in angles_map.keys():
        A_tilda = self.vectors[i].start
        B_tilda = self.vectors[j].start
        OA = Vector(self.O, A_tilda) + self.O_tilda_O
        OB = Vector(self.O, B_tilda) + self.O_tilda_O
        angles_map[(i,j)] = (abs(VectorHelper.get_angle(OA,OB)-90),VectorHelper.
                                        get_angle(OA,OB))

  angles_map = dict(sorted(angles_map.items(), key=lambda item: item[1][0]))
  i,j = list(angles_map.keys())[0]
  return self.vectors[i],self.vectors[j]
```

Функція, що зміщує зображення на визначену кількість пікселів.

```
def shift(self, vertical_shift=0, horizontal_shift=0):
  if vertical_shift >= 0 and horizontal_shift >= 0:
    return Image_Helper(
    img=np.pad(self.img, ((vertical_shift, 0), (horizontal_shift, 0), (0, 0)))[:self.
                                        shape[0],:self.shape[1], :])
  elif vertical_shift < 0 and horizontal_shift >= 0:
    return Image_Helper(
    img=np.pad(self.img, ((0, -vertical_shift), (horizontal_shift, 0), (0, 0)))[-
                                        vertical_shift:,:self.shape[1],:])
  elif vertical_shift >= 0 and horizontal_shift < 0:
    return Image_Helper(
    img=np.pad(self.img, ((vertical_shift, 0), (0, -horizontal_shift), (0, 0)))[:self.
                                        shape[0],-horizontal_shift:,:])
  else:
    return Image_Helper(
    img=np.pad(self.img, ((0, -vertical_shift), (0, -horizontal_shift), (0, 0)))[-
                                        vertical_shift:,-horizontal_shift:, :])
```

Функція, що розбиває зображення на задану кількість рівних частин.

```
def split(self, height_num, width_num):
  """Splits given image into height_num * width_num smaller images"""
  height_length = self.img.shape[0] // height_num
  width_length = self.img.shape[1] // width_num
  return [Image_Helper(img=self.img[i * height_length:(i + 1) * height_length, j *
                                        width_length:(j + 1) * width_length]) for i in
                                        range(height_num) for j in range(width_num)]
```

Функція, що переносить оператор трансформації зображень L в новий базис.

```
def _basys_change(self):
  e1 = (self.O_A.move_to_start())/self.O_A.get_length()
  e2 = (self.O_B.move_to_start())/self.O_B.get_length()
  e_1x,e_1y = e1.get_direction()
  e_2x,e_2y = e2.get_direction()
  self.T = np.array([[e_1x,e_2x],[e_1y,e_2y]])  # from E->e basys
  self.L = self.T@self.L@np.linalg.inv(self.T)  # matrix in basys E_1,E_2
```

Функція, що шукає явний вигляд оператора L шляхом паралельного перенесення базисних осей на кінці векторів перетворення.

```
    def _contsruct_transformation_operator(self):
      e_1 = self.O_A.move_to_start()/self.O_A.get_length()
      e_2 = self.O_B.move_to_start()/self.O_B.get_length()
      O_A1_normalized = (self.O_A1.move_to_start())/self.O_A.get_length()
      O_B1_normalized = (self.O_B1.move_to_start())/self.O_B.get_length()
      line_e1 = VectorHelper.get_line_eq_coeffs(e_1)
      line_e2 = VectorHelper.get_line_eq_coeffs(e_2)

      parallel_e2_line_OA1 = VectorHelper.move_line_to_point(line_e2,O_A1_normalized)
      parallel_e1_line_OA1 = VectorHelper.move_line_to_point(line_e1,O_A1_normalized)
      L_1x_abs = VectorHelper.find_intersection(parallel_e2_line_OA1,line_e1)
      L_1y_abs = VectorHelper.find_intersection(parallel_e1_line_OA1,line_e2)
      projection_vector_L_1x = Vector(e_1.start,SpecialList(L_1x_abs))
      projection_vector_L_1y = Vector(e_2.start,SpecialList(L_1y_abs))
      L_1x = projection_vector_L_1x.get_length()/e_1.get_length()*VectorHelper.
                                          co_direction(e_1,projection_vector_L_1x)
      L_1y = projection_vector_L_1y.get_length()/e_2.get_length()*VectorHelper.
                                          co_direction(e_2,projection_vector_L_1y)

      parallel_e2_line_OB1 = VectorHelper.move_line_to_point(line_e2,O_B1_normalized)
      parallel_e1_line_OB1 = VectorHelper.move_line_to_point(line_e1,O_B1_normalized)
      L_2x_abs = VectorHelper.find_intersection(parallel_e2_line_OB1,line_e1)
      L_2y_abs = VectorHelper.find_intersection(parallel_e1_line_OB1,line_e2)
      projection_vector_L_2x = Vector(e_1.start,SpecialList(L_2x_abs))
      projection_vector_L_2y = Vector(e_2.start,SpecialList(L_2y_abs))
      L_2x = projection_vector_L_2x.get_length()/e_1.get_length()*VectorHelper.
                                          co_direction(e_1,projection_vector_L_2x)
      L_2y = projection_vector_L_2y.get_length()/e_2.get_length()*VectorHelper.
                                          co_direction(e_2,projection_vector_L_2y)

      self.L = np.array([[L_1x,L_2x],[L_1y,L_2y]])
```

Пошук глобального вектору зсуву.

```
  def _find_global_vector(self):
    for i in range(self.split_vert):
      for j in range(self.split_horiz):
        img, shifted_img = self.Images_main_split[j + i * self.split_vert].img, \
        self.Images_shifted_split[j + i * self.split_vert].img

        points = SpecialList(CenterMassFinder(img=img).find_max(self.channel))

        points_shifted = SpecialList(CenterMassFinder(img=shifted_img).find_max(self.
                                          channel))

        # normalizing with the respect to a number of subimages

        points[0] += i * (self.Image_main.shape[0] // self.split_vert)  # coordinates are
                                          (y,x)
        points[1] += j * (self.Image_main.shape[1] // self.split_horiz)

        points_shifted[0] += i * (self.Image_shifted.shape[0] // self.split_vert)  #
                                          coordinates are (y,x)
        points_shifted[1] += j * (self.Image_shifted.shape[1] // self.split_horiz)

        vector = Vector(points, points_shifted)
        self.vectors.append(vector)

        self.global_avg += points
        self.global_avg_shifted += points_shifted

    self.global_avg //= (self.split_vert * self.split_horiz)
    self.global_avg_shifted //= (self.split_vert * self.split_horiz)

    return Vector(self.global_avg,self.global_avg_shifted)
```